

01/18/02
J1062 U.S. PTO

Express Mail : EV049584611US
Date of Deposit: January 18, 2002

1 **IN-KERNEL CONTENT-AWARE SERVICE DIFFERENTIATION**

2 **PRIORITY**

3 This application claims priority from Provisional Application,
4 filed, June 27, 2001, DOCKET NUMBER:YOR920010561US1, assigned Serial
5 Number 60/301,292.

6 **FIELD OF THE INVENTION**

7 This invention is directed to the field of operating systems and
8 computer networking. It is more particularly directed to enable
9 content-aware service differentiation in servers which
10 communicate with clients over a network.

11 **BACKGROUND OF THE INVENTION**

12 Application service providers and Web hosting services that
13 co-host multiple customer sites on the same server cluster or
14 large SMP's are becoming increasingly common in the current
15 Internet infrastructure. The increasing growth of e-commerce on
16 the web means that any server down time that affects the clients
17 being serviced will result in a corresponding loss of revenue.
18 Additionally, the unpredictability of flash crowds can overwhelm
19 a hosting server and bring down multiple customer

1 sites simultaneously, affecting the performance of a large number
2 of clients. It becomes essential, therefore, for hosting
3 services to provide performance isolation, fast recovery times,
4 and continuous operation under overload conditions at least to
5 preferred customers.

6 Each of the co-hosted customers sites may have different
7 quality-of-service (QoS) goals based on the price of the service
8 and the application requirements. Furthermore, each customer
9 site may require different services during overload based on the
10 client's identity (preferred gold client) and the content they
11 access (e.g., a client with a buy order versus a browsing
12 request). When providing service differentiation during overload
13 it is important to know who the request was from and what it is
14 intended to do. The current techniques of using the incoming
15 connections network header values to differentiate requests is
16 not sufficient. The network headers (IP address and port
17 numbers) only identify the origin client machine and the
18 destination client machine and the receiving application at the
19 destination port. With clients behind a proxy all clients will
20 share the same network and cannot be distinguished. Similarly
21 the type of request that can be determined from the port number
22 as being an FTP transfer vs. an HTTP transfer but cannot
23 distinguish a browse order from a buy order. Current commercial
24 switches and routers use a simple threshold-based request discard
25 policy (e.g., a TCP SYN drop mode) to discard the incoming,
26 oldest or any random connection to control overload. Such
27 techniques may delay or control overload but pay a penalty by
28 discarding a high priority gold customer request instead of an
29 ordinary request. These content-unaware approaches are not
30 adequate as they do not distinguish between the individual QoS
31 requirements. For example, a majority of the load is caused by a

1 few CGI requests and most of the bytes transferred belong to a
2 small set of large files. It has been shown in earlier studies
3 that 90% of the web requests are for 10% of the pages at a site.
4 But 10% of the requests account for 80% of the data transferred.

5 Consider for example, a news site with a small size main page
6 which is accessed by a majority of the customers. Few customers
7 will download a large audio/video news segment which will cause a
8 high load on the server and the network. If all the small page
9 requests were dropped they would possibly not reduce the server
10 load as much as the single video segment request. This suggests
11 that targeting specific information and client identities (e.g.,
12 URIs, types of URIs, cookie information, SSL session ids) for
13 service differentiation can have a wide impact during overload.

14 One approach to do content-aware service differentiation is to do
15 it within the application or in user space. Clearly when
16 content-based control is performed it requires that the
17 application is modified and is aware of service differentiation
18 functions during overload. This does not achieve application
19 transparency. Modifying legacy applications is difficult.
20 Secondly, control is handed to the application at a much later
21 stage compared to when the operating system kernel processing
22 begins. In such a case low priority requests or requests that
23 will be discarded subsequently will use precious server resources
24 during overload for no useful work. The service differentiation
25 during overload should follow the "early discard" policy. In
26 this case prioritizing a request or deciding to discard or delay
27 a request should be done as soon as the request is received by
28 the kernel. This implies that an ideal location of content-aware
29 service differentiation is within the operating system kernel.

1 SUMMARY OF THE INVENTION

2 To overcome these problems, the present invention embodies new
3 kernel mechanisms that enable Web servers to provide
4 content-aware service differentiation functions such as policing
5 actions, request dropping, request prioritization, request rate
6 control, and request scheduling, during overload based on client
7 identity and application-specific information. The industry
8 focus on QoS has been network centric but our invention resides
9 on the server allowing for tighter integration with applications
10 and application data like URI's and cookies. Outbound QoS
11 traffic controls alone are not enough to provide the necessary
12 service differentiation required in the current Web server
13 environment. The level of service differentiation required can
14 be provided with integrated outbound and inbound traffic control
15 kernel mechanisms which classify requests based on applicaton
16 content.

17 In this invention we describe methods, systems and apparatus for
18 content-aware service differentiation for overloaded servers
19 within the server operating system. One service-differentiation
20 police action includes silently dropping a communication request
21 received based on the request content. An alternate action is to
22 send a specific message to the client informing that the service
23 is unavailable. In an alternate action the communication request
24 received is scheduled in the kernel based on the request content
25 to determine the order in which the requests are accepted.
26 Different scheduling policies can be specified for this action.
27 In one scheduling policy a priority ordering can be used where a
28 higher priority request is serviced first by the application. A

1 second policy is to use a weighted round-robin ordering where the
2 weights are determined by the service differentiation
3 requirement. In an alternate action the communication request is
4 rate controlled based on the request content. The rate control
5 action limits the rate of new requests entering the system and
6 the number of concurrent requests that are allowed to enter the
7 system. In an alternate action the statistics of the incoming
8 requests are monitored and recorded based on the request content.

9 Thus, the present invention introduces kernel mechanisms in the
10 networking stack of the operating system to support content-aware
11 service differentiation and admission control based on client
12 attributes (IP address, SSL session id, port etc.), server
13 attributes (IP address, type), and the request content (e.g., URI
14 accessed, CGI request information, cookie attributes etc.).

15 An advantage of a kernel-based approach is that it provides lower
16 overhead and better performance for service differentiation as it
17 is placed in the request processing path of the kernel (in the
18 networking stack). It enables "informed early discard" where
19 control can be enforced in the early stages of a request lifetime
20 without consuming system resources but with full knowledge of the
21 request type and client identity. Secondly, it can be
22 implemented easily in any commercial operating system without any
23 complex change to the underlying architectures. Finally, it can
24 be deployed in both the server as well as any external switch or
25 router that controls a cluster of server machines.

26 BRIEF DESCRIPTION OF THE DRAWINGS

1 These and other aspects, features, and advantages of the present
2 invention will become apparent upon further consideration of the
3 following detailed description of the present invention when read
4 in conjunction with the drawing figures, in which:

5 Fig. 1 is a diagram that shows the header format of a standard IP
6 packet, a TCP packet and an HTTP message;

7 Fig. 2 is an example diagram that shows the components of the
8 system architecture of the networking stack for HTTP header based
9 service differentiation, which includes a classifier, an incoming
10 connection queue (also called SYN queue or partial listen queue),
11 an HTTP header classifier, the service differentiation rule table
12 having classification rules and action rules and their processing
13 engine, and a queue of accepted connections (also called accept
14 queue or listen queue), along with the incoming TCP connections
15 and the receiving processes and an external policy agent, in
16 accordance with the present invention;

17 Fig. 3 is an example block diagram that delineates the steps
18 taken by the kernel to execute the associated service
19 differentiation action rule after matching a particular
20 application header-based classification rule in accordance with
21 the present invention;

22 Fig. 4 is an example block diagram that delineates the steps
23 taken by the kernel when the desired action rule is to DROP an
24 incoming connection that matched a particular application
25 header-based rule, in accordance with the present invention;

26 Fig. 5 is an example block diagram that delineates the steps
27 taken by the kernel when the desired action rule is to RATE

CONTROL an incoming connection that matched a particular application header-based rule, in accordance with the present invention;

Fig. 6 is an example block diagram that delineates the steps taken by the kernel when the desired action rule is to SCHEDULE ORDER an incoming connection that matched a particular application header-based rule, in accordance with the present invention;

Fig. 7 is an example diagram that shows the format of the rules tables that include a selector field and the corresponding action, in accordance with the present invention;

Fig. 8 is an example diagram that delineates the different components of the service differentiation module in the kernel, in accordance with the present invention;

and

Fig. 9 is an example diagram that delineates the different components of the policy agent and its communication with the kernel, in accordance with the present invention.

DESCRIPTION OF THE INVENTION

The present invention provides methods, systems and apparatus to efficiently perform content-aware service differentiation in the kernel for overloaded servers based on application layer information. In a first example embodiment of the present invention we leverage the fact that the majority of traffic

1 received by servers from external clients uses the HTTP protocol
2 which in turn is sent over a TCP transport connection. The HTTP
3 header information can be used to identify the type of request,
4 the client identity and other client specific information that
5 can be used to perform informed service differentiation. With
6 the increasing deployment of e-commerce sites, sessions or Web
7 transactions are widely used where web servers exploit cookies in
8 the HTTP header to identify session state. These cookies include
9 attribute value pairs that can also be used to uniquely identify
10 a client and the nature and type of the request. In this
11 embodiment the information in the HTTP headers (URI name or
12 type, cookie attribute-value pairs or other header tags) are used
13 to provide content-aware service differentiation. An alternate
14 embodiment provides *application header-based service*
15 *differentiation*, that enables content-aware service
16 differentiation by examining information in any application
17 header sent over a transport protocol.

18 The service differentiation mechanisms are placed in the kernel's
19 networking stack to transparently intercept the data packets of a
20 new TCP connection to parse the layer headers, classify the
21 attributes in the header based on the classifier rules and find a
22 matching rule and then apply the associated service

23 differentiation action rules that include but are not limited to:

- 24 (i) controlling the rate and burst of new incoming
- 25 requests,
- 26 (ii) dropping (or terminating) a new request,
- 27 (iii) scheduling request accept order (i.e., ordering
- 28 requests based on priority or a weighted round-robin order in the
- 29 accept queue),
- 30 (iv) monitoring and recording request statistics.

1 The classifier involves parsing the (HTTP in this example) header
2 in the kernel and applying an action rule includes waking the
3 sleeping server process only after a decision on how to service
4 the connection is made based on the action rule.

5 Figure 1 shows the formats of a standard HTTP message that
6 includes a IP (101) and TCP (102) packet headers and the HTTP
7 (103) header with the HTTP payload (104). The IP header (101)
8 includes in particular the source and destination IP addresses.
9 The TCP packet header (102) includes the port numbers. The HTTP
10 header (103) includes in particular the HTTP command, the URI
11 being accessed and the list of cookies. The HTTP header is
12 terminated by two CR control characters. The kernel detects the
13 HTTP header semantics by parsing the data stream to detect the
14 delimiter for the header.

15 Figure 2 shows the architecture for application header-based
16 service differentiation. The incoming TCP connection (201) is
17 classified using a SYN classifier (202) and placed in the SYN
18 queue (203) that includes new connections that are not yet
19 established (i.e., the TCP 3-way handshake is not yet been
20 completed). After the TCP handshake completes and data is
21 received to determine the HTTP header (204), the kernel
22 classifies the request based on the header values (206). Such a
23 classification matches the incoming connection values (e.g., URIs
24 and cookies) to a set of classification rules and their
25 associated action rules. These together are called the service
26 differentiation rules are stored in rule table (211) which are
27 populated by a user level policy agent (212) using any standard
28 mechanism to communicate with the kernel (socket or system
29 calls). Once the request is classified the corresponding action
30 rule is applied to it (207). Based on the type of action,

1 further TCP post processing of the request is performed (208) and
2 the request is placed in a queue of accepted connections (209),
3 also called the accept queue or listen queue. The process (210)
4 sleeping on a listen socket is woken up to service the request at
5 the head of the queue.

6 Figure 3 is an example block diagram that delineates the steps
7 taken by the kernel for providing a service differentiation
8 action. The data in the new established TCP connection (301) is
9 parsed to detect the application header and its attributes (302).

10 For the HTTP example, this includes determining the URI and
11 cookies. The URI is the 3rd string in the HTTP header and the
12 cookies start with a cookie delimiter as defined in the HTTP
13 protocol in RFC 2068. The parsed header is then matched against
14 the rule table to find a matching classification rule (303). The
15 associated action rule is found (304) which is then executed
16 (305) and the required processing done (306). After that the
17 standard TCP processing continues (307) and the sleeping server
18 process is then woken to continue processing the data (308).

19 Figure 4 is an example block diagram that delineates the steps
20 taken by the kernel when the matching action rule is a connection
21 "drop". The data in the new established TCP connection (401) is
22 parsed to detect the application header (402). For HTTP this
23 includes determining the URI and cookies. The URI is the 3rd
24 string in the HTTP header and the cookies start with a cookie
25 delimiter as defined in the HTTP protocol in RFC 2068. The
26 parsed header is then matched against the rule table to find a
27 matching classification rule (403). When the associated action
28 rule is to discard the connection (by a DROP action in 404) a TCP
29 RST is sent back to the client (405) and a cleanup is performed
30 (406). Alternately, instead of 405, for HTTP, an HTTP packet is

1 sent back with a return code for the server being busy as defined
2 in the HTTP protocol in RFC 2068.

3 Figure 5 is an example block diagram that delineates the steps
4 taken by the kernel when the matching action rule is a connection
5 "rate control". The data in the new established TCP connection
6 (501) is parsed to detect the application header (502). The
7 parsed header is then matched against the rule table to find a
8 matching classification rule (503). When the associated action
9 rule is to rate limit the connection (by a RATE CONTROL action in
10 504) a compliance check is made against a set of token bucket
11 parameters of (rate and burst) (as shown in 504). The rate
12 control parameters include a rate value which is the rate of new
13 connections that are admitted, and a burst value that is the
14 maximum number of concurrent connections that are admitted. A
15 token bucket with depth equal to the burst parameter and a token
16 regeneration rate set to the rate parameter is used to perform
17 the rate control compliance check. If a token is available in
18 the bucket (508), it is consumed and the connection is inserted
19 in the queue of accepted connections (506) in the desired order.
20 After which the standard processing of the TCP connection
21 continues (507). If a token was not available (508), that is the
22 connection is not compliant, the connection is to be discarded
23 and a TCP RST is sent back to the client (509) and the cleanup is
24 performed (510). Alternately instead of 509, for HTTP an HTTP
25 packet is sent back with a return code for the server being busy
26 as defined in the HTTP protocol in RFC 2068.

27 Figure 6 is an example block diagram that delineates the steps
28 taken by the kernel when the matching action rule is a connection
29 "schedule order". The data in the new established TCP connection
30 (601) is parsed to detect the application header (602). The

1 parsed header is then matched against the rule table to find a
2 matching classification rule (603). When the associated action
3 rule is a schedule ordering (604), the scheduling policy and it's
4 parameters described in the action rule (605) are used to
5 determine the location of the connection in the queue of accepted
6 connections. Any scheduling policy can be used for this purpose.

7 Figure 6 shows the case when the policy is priority-based
8 scheduling. For priority policy, based on the value of the
9 priority assigned to the connection it is inserted in the list of
10 accepted connections (606) in the highest priority first order,
11 where all connections of higher priority are ahead in the queue
12 and all connections with the same priority are inserted in the
13 FCFS (first come first served) order. After which the standard
14 TCP processing continues (607). By ordering connections by
15 priority, a higher priority request (e.g., a buy order) is
16 serviced first by the web server, instead of a lower priority
17 request (e.g., a browse request), as it is placed earlier in the
18 queue of accepted connections. Another policy example is the
19 weighted round-robin scheduler that defines a weight for each
20 class and determines the number of requests that are selected
21 from each class based on the weight and proceeds in a round-robin
22 fashion. The weights are assigned based on an external
23 administrative policy or dynamically by the kernel.

24 Figure 7 shows an example service differentiation rule table
25 which includes a classification rule and action rule pairs. The
26 classification rule could include a URI string (701) with an
27 associated action rule to DROP (701) all connections whose HTTP
28 header included the said URI string. The classification rule
29 could also be a cookie attribute value pair (703) and the action
30 rule including schedule order a priority value (703). In 704,
31 the classification rule includes a URI string with a combined

1 action of rate control along with schedule order with a priority
2 value.

3 Figure 8 shows an example of components of the service
4 differentiation module that includes a parser (802) that parses
5 the application headers from the incoming connection (801) and a
6 classifier (803) that classifies the connection based on the
7 classification rules. The selector (804) finds the associated
8 action rule and the performer executes the desired action.

9 Figure 9 shows the interaction between the user space policy
10 agent (901) and the kernel components. The communicator (902)
11 uses an API to communicate with the policy agent. The
12 initialiser (903) sets up the service differentiation rule table
13 (905) and initializes the rules. The manager (904) adds new
14 rules and deletes and updates existing rules based on the policy
15 agent commands.

16 Thus the invention also includes a method comprising forming a
17 rule. The step of forming often includes the steps of:
18 communicating from a user space to a kernel with an application
19 interface; instantiating service differentiation rules for an
20 application tag within the kernel which include classification
21 and action rules; and deleting and adding rules based upon a user
22 request. In some embodiments the method further includes
23 updating rules based upon a user request.

24 The present invention can be realized in hardware, software, or a
25 combination of hardware and software. A visualization tool
26 according to the present invention can be realized in a
27 centralized fashion in one computer system, or in a distributed
28 fashion where different elements are spread across several

1 interconnected computer systems. Any kind of computer system -
2 or other apparatus adapted for carrying out the methods and/or
3 functions described herein - is suitable. A typical combination
4 of hardware and software could be a general purpose computer
5 system with a computer program that, when being loaded and
6 executed, controls the computer system such that it carries out
7 the methods described herein. The present invention can also be
8 embedded in a computer program product, which comprises all the
9 features enabling the implementation of the methods described
10 herein, and which - when loaded in a computer system - is able to
11 carry out these methods.

12 Computer program means or computer program in the present context
13 include any expression, in any language, code or notation, of a
14 set of instructions intended to cause a system having an
15 information processing capability to perform a particular
16 function either directly or after conversion to another language,
17 code or notation, and/or reproduction in a different material
18 form.

19 Thus the present invention includes an article of manufacture
20 which comprises a computer usable medium having computer readable
21 program code means embodied therein for causing a function
22 described above. The computer readable program code means in the
23 article of manufacture comprises computer readable program code
24 means for causing a computer to effect the steps of a method of
25 this invention. Similarly, the present invention may be
26 implemented as a computer program product comprising a computer
27 usable medium having computer readable program code means
28 embodied therein for causing a a function described above. The
29 computer readable program code means in the computer program
30 product comprising computer readable program code means for

1 causing a computer to effect one or more functions of this
2 invention. Furthermore, the present invention may be implemented
3 as a program storage device readable by machine, tangibly
4 embodying a program of instructions executable by the machine to
5 perform method steps for causing one or more functions of this
6 invention.

7 It is noted that the foregoing has outlined some of the more
8 pertinent objects and embodiments of the present invention. This
9 invention may be used for many applications. Thus, although the
10 description is made for particular arrangements and methods, the
11 intent and concept of the present invention is suitable and
12 applicable to other arrangements and applications. For example,
13 although the description is with regard to HTTP on top of TCP/IP,
14 the concepts of the present invention may similarly be employed
15 for other protocols. It will be clear to those skilled in the
16 art that modifications to the disclosed embodiments can be
17 effected without departing from the spirit and scope of the
18 present invention. The described embodiments ought to be
19 construed to be merely illustrative of some of the more prominent
20 features and applications of the present invention. Other
21 beneficial results can be realized by applying the disclosed
22 invention in a different manner or modifying the present
23 invention in ways known to those familiar with the art.